In C++, how can I make a default parameter be the this pointer of the caller?

devblogs.microsoft.com/oldnewthing/20231207-00

December 7, 2023

Raymond Chen

Consider the following class:

```
struct Property
{
    Property(char const* name, int initial, Object* owner) :
        m_name(name), m_value(initial), m_owner(owner) {}

    ⟦ other methods elided - use your imagination ⟧

    char const* m_name;
    Object* m_owner;
    int m_value;
};
```

Suppose the idea is that you have a class that has a bunch of properties as members, and the containing class serves as the owner.

```
struct Widget : Object
{
    Property Height{ "Height", 10, this };
    Property Width{ "Width", 10, this };
};
```

Now, it's a bit annoying having to say `this` each time you define a property. Is there some way that the `Property` constructor can infer the class that is creating it?

You can't make a member function default parameter dependent upon its own `this`, but what about making it dependent on the caller's `this`?

No, that doesn't work either. Default parameters are resolved at the point of declaration, not at the point of invocation.

```
int v;

namespace N
{
    int v;

    struct Example
    {
        Example(int value = v);
    };

}

void test()
{
    int v;
    N::Example e; // which "v" does this use?
}
```

The answer is that it uses `N::v`, because that is the `v` that is found at the point the `int value = v` is encountered. The local variable `v` is not considered because it is not in scope, and the global variable `::v` is not considered because it has been shadowed by `N::v`. It is irrelevant that `::v` and the local variable `v` are in scope and visible at the time the constructor is called.[1]

But again, all is not lost. We just have to find another trick.

```
struct Widget : Object
{
    Property Prop(char const* name, int initial)
    { return Property(name, initial, this); }

    Property Height{ Prop("Height", 10) };
    Property Width{ Prop("Width", 10) };
};
```

The trick is to declare a helper method inside `Widget` that creates the `Property` and adds the `Widget`'s `this` pointer as the final parameter.

You can make this reusable by factoring it into a helper base class that uses the curiously recurring template pattern (commonly known as CRTP).

```
template<typename D>
struct PropertyHelper
{
    Property Prop(char const* name, int initial)
    { return Property(name, initial, static_cast<D*>(this)); }
};

struct Widget : Object, PropertyHelper<Widget>
{
    Property Height{ Prop("Height", 10) };
    Property Width{ Prop("Width", 10) };
};
```

If you have access to C++23's deducing this, then you can simplify it further:

```
struct PropertyHelper
{
    template<typename Parent>
    Property Prop(this Parent&& parent, char const* name, int initial)
    { return Property(name, initial, &parent); }
};

struct Widget : Object, PropertyHelper
{
    Property Height{ Prop("Height", 10) };
    Property Width{ Prop("Width", 10) };
};
```

You can go even further and just put `Prop()` in the `Object`.

```
struct Object
{
    ⟦ other methods elided - use your imagination ⟧

    template<typename Parent>
    Property Prop(this Parent&& parent, char const* name, int initial)
    { return Property(name, initial, &parent); }

};

struct Widget : Object
{
    Property Height{ Prop("Height", 10) };
    Property Width{ Prop("Width", 10) };
};
```

Is this an improvement over typing `this` repeatedly? I'm not sure.

[1] If we had moved the declaration of `N::v` to after the definition of `struct Example`, then the `int value = v` would have resolved to `::v`, since `N::v` hasn't been declared yet.